REMARKS

Reconsideration of this application as amended is requested. By this amendment Applicants have amended independent claims 1 and 11 for clarity. Claims 1-11 remain in the case.

The Examiner rejected claims 1, 6, 7 and 8-10 on the grounds of nonstatutory obviousness-type double patenting as being unpatentable over claims 1-3 and 5-7 of co-pending U.S. Patent Application Serial No. 09/776040; rejected claims 1, 6, 7 and 9-11 under 35 U.S.C. 102(b) as being anticipated by Swift et al (WO 98/57268 – "Swift"); rejected claims 2-5 and 8 under 35 U.S.C. 103(a) as being unpatentable over Swift in view of Matsui (U.S. Patent No. 6,560,723) or Gessel et al (U.S. Patent No. 5,732,213 – "Gessel").

Double Patent Rejection

With respect to the double patenting rejection, the Examiner indicated that such rejection is provisional because the conflicting claims have not in fact been allowed. Therefore upon the allowance of patentable claims, Applicants will submit a terminal disclaimer in compliance with 37 C.F.R. 1.321(c) to overcome the double patenting rejection, as both applications are commonly owned.

§103 Rejection

In contradistinction to Applicant's claimed invention Swift discloses a multi-protocol message sequence generator that enables a user to define a sequence of messages and transmit the messages to a target network object for testing, which target object is a network management system that is responsive to the reception of the message sequence. The message sequences correspond to actual message sequences transmitted by network source objects to target objects in a production network. Network sources correspond to switches, routers, bridges, repeaters, etc. – any device capable of communicating messages on a network. A graphical user interface (GUI) is provided that allows the user to simply select the message type, content and sequencing the user wishes to generate. The user may also edit existing messages. As a result the user has a wide variety of options for creating testing scenarios in a quick, easy and efficient manner.

As shown in Fig. 1, a sequence generator **102** is connected to a communications network **103** to which also is connected a data collector **108**. One or more store forward files **110** are accessible by the data collector. Each store forward file is monitored by a data distributor **112** which passes on a message sequence **106** to a target object **114**. Fig. 2 shows a message sequence generator **102** that includes a message sequence engine **218** that communicates with a GUI **212**, a message database **214** (containing actual messages previously sent) and a message text file **216** (similar to message database, but creatable using standard word processing) and provides the message sequence via a network interface **224** to the communications network, preferably a TCP/IP network, i.e., a network that uses the transmission control protocol for robustness of data transmission and the internet protocol for transmitting data from location to location or node to node. The message sequence engine allows the user to create a message sequence definition **222** via the GUI. The message sequence definition defines the message sequence.

As with all other communications protocol, TCP/IP is composed of layers:

- **IP** - is responsible for moving packets of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.
- **TCP** - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.

It is noted that Swift does not discuss emulation of a protocol layer and, aside from the reference to the TCP/IP network as the transmission medium, does not mention any protocol layers at all. The TCP/IP network is merely the medium for transferring the message sequences from the network sources (message generator) to the target objects. The purpose of the Swift invention is to test

target objects, not to test any specified protocol layer.

The Examiner argues that Swift teaches a method of setting up a communication procedure between instances that include selecting the instances that take part in the communication procedure, one instance being a protocol tester and another instance being an item under test, citing page 1, paragraph 3, lines 1-9; page 6 paragraph 3; and page 8, paragraph 5 – A network management system receives "events (messages) from a wide variety of network components, such as network switches and network routers" to which the management system responds in a specific way to certain of the events. It is apparent that the Examiner is equating the message sequence generator of Swift to Applicant's claimed protocol tester and the target object to the claimed item under test.

The Examiner then states that Swift teaches selecting a protocol layer to be emulated by the protocol tester for testing a specified protocol layer of the item under test on the basis of the communication procedure, citing the page 7, paragraph 1, line 1 - paragraph 2, line 9; page 6, paragraph 3; and page 8 paragraph 5 – the use of the TCP or IP or other protocol capable of transferring messages. However Swift is not emulating any of these protocols as recited by Applicant, but is merely using these protocols as a transmission medium for getting the message sequences from the message sequence generator to the target object. There is no indication in Swift that what is being tested is "a specified protocol layer of the item under test" as recited by Applicant. Swift does not emulate a protocol layer, and therefore does not "select a protocol layer" as recited by Applicants. Swift merely indicates that in the particular embodiment using the TCP/IP communication network the message sequence engine produces TCP/IP capable applications, i.e., message sequences that are transmittable over the TCP/IP network.

The Examiner then states that Swift teaches selecting abstract communication interfaces of the emulated protocol layer for the communication procedure, citing page 7, paragraph 2, lines 1-9 – software applications that build interfaces. However Applicant finds no reference in the cited portion of Swift to building interfaces, especially "abstract communication interfaces of the emulated protocol layer" as recited by Applicants. The only reference in the cited paragraph is how the message sequence engine is implemented as a software application using a fourth generation language for developing windows graphically.

The Examiner further states that Swift teaches selecting communication data contained in description files to be exchanged at the abstract communication interfaces, citing page 9, paragraph 2. The cited paragraph indicates how a Swift user builds a message sequence definition by inputting a sequence name and having the message sequence engine load the previously saved sequence definition from the message database to the message sequence definition. It appears that the Examiner is equating the claimed description files to the message database of Swift and the claimed communication data to the saved sequence definition. However these are message sequences, not communication data that are "exchanged at the abstract communication interfaces" as recited by Applicant.

Lastly, the Examiner states that Swift teaches automatically setting up through the protocol tester the communication procedure on the basis of the selections made in the above selecting steps, with parameters for the abstract communication interfaces and the communication data selecting steps being made graphically, citing page 7, paragraph 3, lines 1-5 and Fig. 4A, items 406-422 – message created, interfaces produced with PowerBuilder/PowerSockets, specific description file in Fig. 3 (message sequence definition 222). Applicant does not find any reference in the cited portion of Swift to automatic setting up through the message sequence generator of the communication procedure on the basis of the selections made, as the message sequence generator merely builds the message sequence by the user interacting graphically with the message sequence engine. The message sequence engine allows the user to create the message sequence definition

and then upon the user's request to transmit the message sequence corresponding to the message sequence definition onto the network. Applicant can only assume, absent a clear statement by the Examiner, that the Examiner is equating the transmission of the message sequence corresponding to the message sequence definition to the claimed automatic setting up of the communication procedure (see Annex A). Applicant submits that there is insufficient information in Swift to arrive at such a conclusion.

Finally, Applicant recites that a message within the communication data is defined graphically, the message containing a variable so that the other instance performs one of several activities as a function of the content of the variable. There is no indication in Swift of the graphical definition of a variable that controls the other instance or device under test. Rather Swift is an event based system where the device under test responds to an event, rather than a variable within a received message as recited by Applicant.

Therefore claims 1 and 11, which have been amended to clarify that the protocol selection step is the selection of a protocol *to be emulated*, are deemed not to be anticipated by Swift since Swift neither teaches or suggests to one of ordinary skill in the art the steps of selecting a protocol layer to be emulated, selecting the abstract communication interfaces for the selected protocol layer, selecting the communication data for exchange across the abstract communication interfaces, the automatic setting up of the communication procedure based upon selections made, nor defining the message with a variable that determines activities that the device under test performs. Swift is deemed merely to generate message sequences for target objects without selecting any particular protocol layer to be emulated, and uses an established communication procedure rather than setting up the communication procedure based upon the protocol layer selections. Nowhere in Swift is there any reference to any terminology that indicates testing of a selected protocol layer, i.e., there is no reference to protocol layers or the OSI model, there is no reference to abstract communication interfaces, there is no reference to service access points, there is no reference to protocol data units and there is no reference to abstract service primitives. Finally there is no mention in Smith of

defining a message with a variable that controls activities performed by the device under test. It appears that the Examiner is merely assuming, or taking Official Notice, of these items although he cites no appropriate reference or indicates how they actually fit within the message sequence generator of Swift.

Conclusion

In view of the foregoing amendment and remarks allowance of amended claims 1 and 11 together with claims 2-10 dependent therefrom is urged, and such action and the issuance of this case are requested.

Respectfully submitted,
Christian Zander

By /Matthew D. Rabdau/
Matthew D. Rabdau
Reg. No. 43,026
Attorney for Applicants

TEKTRONIX, INC.
P. O. Box 500, MS 50-LAW
Beaverton, Oregon 97077
(503) 627-7261

7468 US